Christopher Prempeh

CS-300 DSA: Analysis and Design

October 2025

Project One: Pseudocode and Runtime Analysis

## Vector Data Structure Pseudocode

STRUCT Course  
 courseNumber  
 name  
 prerequisites = list of courseNumbers  
  
FUNCTION loadCourses(filename)  
 open file  
 for each line in file  
 tokens = split line by comma  
 if tokens length < 2  
 print "Format error: missing course number or name"  
 continue  
 course = new Course  
 course.courseNumber = tokens[0]  
 course.name = tokens[1]  
 for each token from index 2 to end  
 if token exists as a courseNumber in the file  
 add token to course.prerequisites  
 else  
 print "Format error: prerequisite not found"  
 add course to coursesVector  
 close file  
 return coursesVector  
  
FUNCTION searchCourse(coursesVector, courseNumber)  
 for each course in coursesVector  
 if course.courseNumber == courseNumber  
 print course.courseNumber + ", " + course.name  
 if course.prerequisites not empty  
 print "Prerequisites:"  
 for each prereq in course.prerequisites  
 print prereq  
 return  
 print "Course not found"  
  
FUNCTION printAllCoursesSorted(coursesVector)  
 sort coursesVector by courseNumber ascending  
 for each course in coursesVector  
 print course.courseNumber + ", " + course.name

## Hash Table Data Structure Pseudocode

FUNCTION loadBids(path)  
 open CSV file  
 for each row in file  
 parse fields into Bid  
 Insert(Bid)  
  
FUNCTION Insert(Bid)  
 key = convert bidId to int  
 index = hash(key)  
 if nodes[index] empty  
 store bid  
 else  
 append new node to chain  
  
FUNCTION Search(bidId)  
 key = convert bidId to int  
 index = hash(key)  
 node = nodes[index]  
 while node not null  
 if node.bidId == bidId  
 return bid  
 node = node.next  
 return not found  
  
FUNCTION Remove(bidId)  
 key = convert bidId to int  
 index = hash(key)  
 node = nodes[index]  
 if head matches  
 replace/reset head  
 else  
 traverse chain, unlink match  
  
FUNCTION PrintAll()  
 for each bucket  
 print head if valid  
 traverse chain printing each

## Binary Search Tree Data Structure Pseudocode

STRUCT Course  
 string courseNumber  
 string name  
 list<string> prerequisites  
  
STRUCT Node  
 Course data  
 Node\* left  
 Node\* right  
  
CLASS CourseBST  
 Node\* root  
  
 METHOD insert(course)  
 if root is null  
 root = new Node(course)  
 else  
 call insertNode(root, course)  
  
 METHOD insertNode(node, course)  
 if course.courseNumber < node.data.courseNumber  
 if node.left is null  
 node.left = new Node(course)  
 else  
 insertNode(node.left, course)  
 else  
 if node.right is null  
 node.right = new Node(course)  
 else  
 insertNode(node.right, course)  
  
 METHOD find(courseNumber)  
 cur = root  
 while cur != null  
 if courseNumber == cur.data.courseNumber  
 return cur.data  
 else if courseNumber < cur.data.courseNumber  
 cur = cur.left  
 else  
 cur = cur.right  
 return NULL  
  
 METHOD inOrderTraversePrint()  
 call inOrder(root)  
  
 METHOD inOrder(node)  
 if node == null, return  
 inOrder(node.left)  
 printCourse(node.data)  
 inOrder(node.right)  
  
FUNCTION printCourse(course)  
 print course.courseNumber + ', ' + course.name  
 if course.prerequisites is empty  
 print 'Prerequisites: none'  
 else  
 print 'Prerequisites: '  
 for each p in course.prerequisites  
 print p on same line separated by commas

## Menu Pseudocode

PRINT "Welcome to the ABCU Course Advising System"  
DO  
 PRINT "1. Load course data"  
 PRINT "2. Print all courses (sorted)"  
 PRINT "3. Print course details"  
 PRINT "9. Exit"  
 INPUT choice  
  
 IF choice == 1  
 loadCourses(filename)  
 ELSE IF choice == 2  
 printAllCoursesSorted()  
 ELSE IF choice == 3  
 INPUT courseNumber  
 searchCourse(courseNumber)  
 ELSE IF choice == 9  
 PRINT "Goodbye!"  
 ELSE  
 PRINT "Invalid option"  
WHILE choice != 9

## Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Vector | Hash Table | Binary Search Tree |
| File Load | O(n) | O(n) | O(n log n) |
| Search | O(n) | O(1) average / O(n) worst | O(log n) average / O(n) worst |
| Insert | O(1) | O(1) average / O(n) worst | O(log n) average / O(n) worst |
| Sort/Traverse | O(n log n) | N/A (unsorted) | O(n) |

## Advantages, Disadvantages, and Recommendation

## The vector structure is simple and effective for small datasets. It offers predictable indexing and is easy to sort, but searching or inserting in the middle of a large dataset can be slower (O(n)). The hash table provides quick average-time search and insertion (O(1)), making it ideal for fast lookups, although it does not maintain sorted order and can perform poorly if many collisions happen. The binary search tree (BST) stores data in sorted order efficiently, enabling O(log n) searches and insertions when balanced, but unbalanced trees can degrade to O(n). Based on the advisor’s needs—searching for specific courses and printing all courses in alphanumeric order—the BST is the best choice. It naturally keeps data sorted and allows for efficient retrieval and printing. Therefore, the BST structure is recommended for the implementation phase.